Daniel Imberman
Scott Cesar

# Implementation of Berkeley Coreference Resolver

## Abstract

Over the course of this quarter we have taken on the task of implementing a system of coreference resolution described in the paper Easy Victories and Uphill Battles in Coreference Resolution. Implementing the system in Scala, we were able to create a coreference resolver that parsed input written in the format used by the ConLL Shared Task (ConLL). While we were unable to create a system that matched the accuracy described in the Berkeley paper, we were able to beat the performance of a randomized weight vector, indicting our system manages to do some learning.

## Easy Victories and Uphill Battles

The source of our term project was a paper released by Greg Durret and Dan Klein of UC Berkeley in 2013. Durret and Klein created a co-reference resolution system that abandoned Stanford's sieve based system for a machine learning system that was simultaneously more accurate and less engineering intensive. Using a small group of surface level feature templates, the Berkeley team was able to implement a system that could implicitly infer many of the more nuanced linguistic rules that Stanford engineers explicitly placed into their system.

Implementation of this system would be dependent on three main components. First, a data parser will need to be developed to extract the SURFACE features from the CoNLL Shared Task corpus. Second a feature extractor will take information extracted from the SURFACE feature set and construct a feature vector, which will contain features specific to potential co-references in hopes that future matches will increase the likelihood of a correct link. Finally, we will need to compute a weighting vector which indicates the correlation of features to weather or not something is corefferential.

**Data Retrieval**

This project relies on acquiring both the ConLL shared task corpus and the OntoNotes Corpus v. 4.0. Eventually through direct contact with University of Pennsylvania's Linguistic Data Consortium we were able to retrieve the corpus need to complete this project.

**Data Parsing**

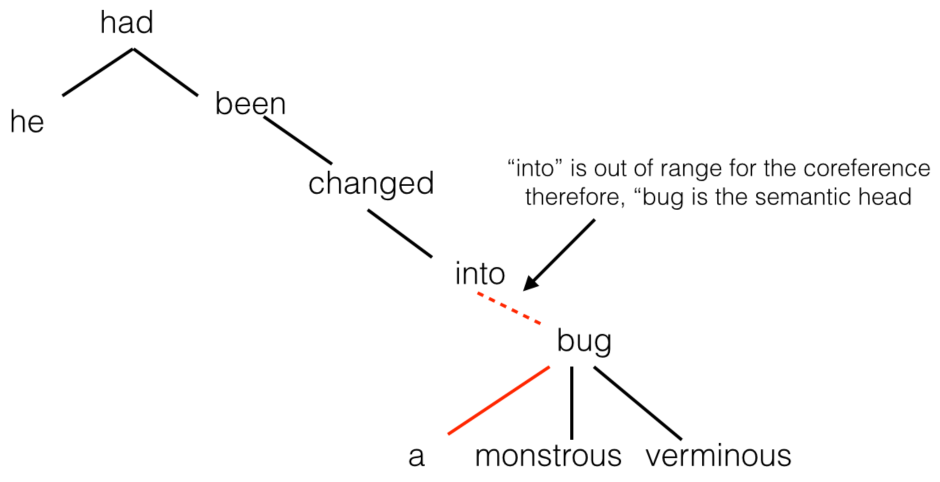
The ConLL dataset essentially worked on top of the OntoNotes Corpus to richly annotate and preprocess data for the purpose of co-reference resolution. On top of the words and a clustering of coreference mappings we were given:

1. Part of speech.
2. A Penn tree showing the dependency structure of each sentence.
3. Predicate lemma
4. Each word's speaker
5. A mention ID that could be used to cluster a co-reference back to its entity.

With this parsing structure, many of the SURFACE features were trivial to retrieve, with the exception of the headword.

By implementing Stanford's CoreNLP data parser, we were able to construct a dependency tree for each sentence. However, there was an issue in that the program was only able to implement the tree if the entire tree was intact. Since coreferences are limited to small subsections of the sentence, we had to find a way to dynamically guess the headword for any phrase within a sentence.

We accomplished this by implementing a dependency HashMap. We implemented a HashMap that would allow use to input an index, and retrieve the index of its parent node. Once we had this, we could say that if at any point a node either pointed to itself (meaning it was the sentence head), or pointed to an index outside of the phrases word range (meaning that it was above all other words in the phrase), then the current word was the semantic head.


had
he
been
changed
into
bug
a
monstrous
verminous
"into" is out of range for the coreference therefore, "bug is the semantic head

*Given the phrase "he had been changed into a monstrous verminous bug, and the coreference phrase "a monstrous verminous bug", our system will correctly identify "bug" as the semantic head of the statement.*

## **Feature Extraction**

The point of the feature extraction step was to create a gigantic vector that take an individual feature and describe whether it relates to an anaphoric or non-anaphoric pair. Imagine we have the statements "Bill Clinton was the president of the USA. In 1996, the president dealt with a gridlocked congress and calls for his impeachment." In this pair, the semantic heads would be inserted into the vector as “anaphor=true hw=the president” and “anthw=Bill Clinton”. On their own, these features mean nothing, but if during the training phase we find a large number of cases where a current head of “the president” and an antecedent head of “Bill Clinton” correlate with anaphoricity, then its weight will be increased and all future cases of a pair of phrases that have these head words will be more likely to be considered anaphoric.

Towards the end of our project, we realized that the Berkeley researchers had done a certain level of pronoun parsing to increase the accuracy of their results. With this in mind we were able to find the pronoun dictionary within their code, and having realized that their pronoun dictionary was trivial in implementation, and not important to the core algorithm, we decided to simply use theirs rather than waste our time building one for ourselves.

## Coreference Chain

Coreferences are stored as a chain where any given mention links to any previous mention, which determines that two elements are coreferential. If an element is determined to be coreferenced to itself then it starts a new coreference group. This is analogous to the CONLL shared task, which ask that coreferences be put into groups, the group in our work being the series of all coreferences whose root is determined to be coreference to itself.

## Classification

The coreference chain means that for each document we need to determine a single chain of entries $\boldsymbol{a}$ such that $a_i=j$ specifies that the ith mention is anaphoric to the jth mention. To determine this, we use a loglinear model for the probability of a particular coreference chain ***a*** being explained by a particular document ***x***.

$$P(a|x) \propto \exp\left(\sum_{i=1}^{n} \mathbf{w}^\top \mathbf{f}(i, a_i, x)\right)$$

As was mentioned in the paper, ***log(P(a|x))*** decomposes into a linear combination. This means that we can maximize the probability by maximizing each individual component.

## Training

Our training function is an implementation of softmax-margin optimized by adagrad on a log likelihood function of P(a|x). The essence of this likelihood function is that it determines P($a_{x(i)}$| weights): the chance that we can explain the optimal assignment of a coreference chain with a given weight vector. In other words, it gives us a singular value to represent the quality of a particular set of weights. For each document we compute the marginal probability that each potential coreference assignment correct, and the marginal probability that each consistent gold coreference assignment is correct. In this context, consistent gold coreference assignments are mentions that are members of the same gold cluster, excluding self-reference unless ***$a_i$*** is the root of a new cluster. Each of these marginal scores is modified by a boosting cost function based on whether or not the particular coreference assignment is wrong. Given this information we can then calculate the gradient of the weight vector as the gold consistent marginals minus the generally computed marginal scores. We can then use the gradient to solve the maximization problem to find the weight vector which has the highest likelihood of correctly determining the coreference chain ***a***. The boosting loss function used was based on whether a particular assignment was a Wrong Link (WL), indicating it was put into the wrong cluster, a False New (FN) indicating it was made a cluster root when it shouldn't have been, or a False Anaphor (FA) indicating it should have been a cluster root but was not marked as a cluster root. The penalty given to each type of incorrect answer was defined in the paper:.1 for FA, 3.0 for FN, and 1.0 for WL.

An explicit formulation of the likelihood function is:

$$\ell(\mathbf{w}) = \sum_{k=1}^{t} \log \left( \sum_{a \in \mathcal{A}(C_k^*)} P'(a|x_k) \right) + \lambda \|\mathbf{w}\|_1$$

**Adagrad Gradient Descent**

Adagrad Gradient descent is a technique for minimizing or maximizing a function where some different dimensions of the gradient have wildly varying importance. Specifically, by calculating the rate of change of each part of the gradient and adjusting the learning rate for that part accordingly, we can capture the nature of the search space by allowing features which show up rarely but strongly affect the results to have a higher impact.

**Findings**

In the end we were actually able to match the findings of the Berkeley paper. On average our coreference resolver had between a 61-63% accuracy rate, which while slightly less than the Berkeley paper's 66%, is still very good considering that there were parsing aspects that were implemented by the Berkeley team that were not explicitly discussed in their paper.

Daniel Imberman
Scott Cesar

Implementation of Berkeley Coreference Resolver

## Differences In Implementation

While we were able to reproduce the results of the Berkeley paper, there was a significant aspect of their implementation that they failed to explicitly mention in their paper: coreference identification. In their code, the Berkeley researchers created a way to parse the entire document, and attempt to learn what kinds of phrases constitute coreferences and what kinds do not. Because we were doing this project in one quarter, and this problem was not important to the core elements of their findings, we decided to skip this part of the implementation and elected to instead take advantage of the annotations in the CoNLL files to determine which phrases could be considered as mentions.